Java RMI programming by example

Chris Greenhalgh, University of Nottingham, School of Computer Science and IT

Mar 2008. For Java 1.1+

Note: variables which you need to replace with your own values are shown as <*variable*>, e.g. <*portnumber*>, <*your-username*>.
Note: command you type are shown in bold.

Note that your shell environment must be configured to let you use Java. E.g.

- on the CSiT UNIX servers you may need to add "JDK" to the "PACKAGES" list in the file ".cshrc" in your home directory and log in again (or 'source ~/.cshrc').
- on the CS terminal room PCs you may need to set the following shell variables in each shell you use (if there is no C:\jdk1.5 then look for another installation of the a Java Development Kit, e.g. in Program Files\Java and use that path instead):

```
set JAVA_HOME=C:\jdk1.5
set PATH=%PATH%;%JAVA_HOME%\bin
set CLASSPATH=.
```

1. Define the remote interface

Define the interface by which the remote object will be accessed. It must extend the interface java.rmi.Remote and all methods must be declared to throw java.rmi.RemoteException (plus whatever else they might throw).

E.g., a ticket server, which returns a new ticket number on each query (like at the delicatesan counter of a supermarket). The name argument is just to show it works.

--------TicketServer.java--------

```
// TicketServer.java

import java.rmi.*;

public interface TicketServer extends Remote
{
  public int getNextTicket(String name) throws RemoteException;
}
```

--------end of TicketServer.java--------

2. Write the server

Write the implementation of your service. This must be a subclass of (extend) `java.rmi.server.UnicastRemoteObject` and must implement your remote interface (defined in the

previous step). It can include instance variables. In our example we will also include a static main method to make a complete application which creates and registers a single instance of our server.

--------TicketServerImpl.java--------

```
// TicketServerImpl.java

import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class TicketServerImpl extends UnicastRemoteObject
          implements TicketServer
{
  int nextTicket=0;
  TicketServerImpl() throws RemoteException { };

  public int getNextTicket(String name) throws RemoteException {
    System.out.println("Issue a new ticket for " + name);
    return nextTicket++;
  }

  public static void main(String [] args) {
    // install RMI security manager
    System.setSecurityManager(new RMISecurityManager());

    // arg. 0 = rmi url
    if (args.length!=1) {
      System.err.println("Usage: TicketServerImpl <server-rmi-url>");
      System.exit(-1);
    }

    try {
      // name with which we can find it = user name
      String name = args[0];
      //create new instance
      TicketServerImpl server = new TicketServerImpl();
      // register with nameserver
      Naming.rebind(name, server);
      System.out.println("Started TicketServer, registered as " + name);
    }
    catch(Exception e) {
      System.out.println("Caught exception while registering: " + e);
      System.exit(-1);
    }
  }
}
```

--------end of TicketServerImpl.java--------

3. Compile

Compile your classes (without errors!) (UNIX and Windows).

```
> javac TicketServer.java TicketServerImpl.java
```

4. Make stubs

Process with the java RMI compiler to make client stubs and server skeleton (UNIX and Windows).

```
> rmic TicketServerImpl
```

5. Write client

Create a client to use the server.

-------TicketClient.java-------

```
// TicketClient.java

import java.rmi.*;

public class TicketClient {
  public static void main(String [] args) {
    // install RMI security manager
    System.setSecurityManager(new RMISecurityManager());

    // arg. 0 = name of server
    if (args.length!=1) {
      System.err.println("Usage: TicketClient <server-rmi-url>");
      System.exit(-1);
    }

    // look up in nameserver
    String fullname = args[0];
    TicketServer server = null;
    try {
      server = (TicketServer)Naming.lookup(fullname);
    } catch (Exception e) {
      System.out.println("Caught an exception doing name lookup on "+fullname
                         +": "+e);
      System.exit(-1);
    }

    // get ticket - remote method invocation!
    try {
      int ticket = server.getNextTicket("TicketClient");
      System.out.println("Got ticket " + ticket);
    } catch (Exception e) {
      System.out.println("Exception caught while getting ticket: "+e);
      System.exit(-1);
    }
  }
}
```

-------end of TicketClient.java-------

6. Compile

Compile the client (without errors) (UNIX and Windows).

```
> javac TicketClient.java
```

7. Publish client stubs

Copy the .class files needed by the client and registry (i.e. the remote interface definition and the client stub code) to a place which is accessible by the rmiregistry and by the client when it is run, e.g. in your web space (on UNIX):

```
> cp TicketServer.class TicketServerImpl_Stub.class ~/public_html/
```

(This is on CSiT UNIX machines; it will be different from the PC, e.g. use the windows explorer.)

8. Start registry

Check that the name server (registry) is running (UNIX only):

```
> ps -auxww | grep rmiregistry | egrep -v grep
```

[or ps -df ... if you have that sort of ps]

If you are on a PC then you can be pretty sure that it will only be running if you have just started it. Use the task manager if you need to check.

Note, If it appears to be but you're program (see later) fails anyway because of:

```
  java.net.NoRouteToHostException: connect timed out: <port-num>
```

then it must be running on a different port, so start your own anyway...

If the rmiregistry is not running (or on an unknown port) then start it in a new shell and leave it running... (UNIX and Windows)

```
> rmiregistry <port-number>
```

and leave it running in that shell while you run the server and client(s) (but remember to kill it before you go). On UNIX, make up a port number at random between 5000 and 32767; if it gives an error try a different number. On Windows you can omit the port and use the default (1099) if it is not already in use. Note that the rmiregistry **does not print any messages** at all (this does not mean it is broken).
NB: do not run the registry in the directory which contains your class files, or downloading of stubs will not work properly. E.g. run in an empty directory.

NOTE: (at least in Java 1.1) if you change the definition of an RMI interface (e.g. change methods, arguments or return types) which is registered with the registry then you will need to restart the registry before you can use the new version. Otherwise the old class will still be loaded in the registry, which will then complain about class versions not matching when you try to register the new version.

9. Run server

For Java 1.2+ / Java 2: make sure have a suitable security policy file in the current directory, e.g. java.policy: (NOT advised for code you don't trust...!)

```
grant {
    // anything
    permission java.security.AllPermission;
};
```

In a new shell run the server, specifying (to the registry and client) where the client .class file can be obtained (UNIX and windows, as one command):

```
> java -Djava.rmi.server.codebase=http://www.cs.nott.ac.uk/~<your-username>/
-Djava.security.policy=java.policy TicketServerImpl rmi://:<port-number>/<a-name>
```

It should print something like:

```
 Started TicketServer, registered as rmi//:<port-number>/<a-name>
```

If you are using a version of Java prior to Java 1.2 (Java 2) then you will not need to specify the java.security.policy value.

If the rmiregistry and/or the client are not be able to load the rmic-generated ..._Stub.class file(s) or the remote interface .class file(s) using their own CLASSPATH or the specified codebase then you will get an error like:

```
 Caught exception while registering: java.rmi.UnexpectedException:
 Unexpected exception; nested exception is:
         java.lang.ClassNotFoundException: TicketServerImpl_Stub
```

you should check that these .class files are accessible as specified so that the registry and client(s) can use it to load the class files.

10. Run client

For Java 1.2 / Java 2: make sure have a suitable security policy file in the current directory, e.g. java.policy (as above).

In another shell now run the client a few times to check it works (UNIX and Windows). The <*machine-name*> and <*port-number*> must match the RMI registry used by the server, and <*a-name*> must match the name used by the server in that registry. (all one command):

```
> java -Djava.security.policy=java.policy TicketClient
    rmi://<machine-name>:<port-number>/<a-name>
```

Again, if you are using a version of Java prior to Java 1.2 (Java 2) then you will not need to specify the java.security.policy value.
You can copy TicketClient.class, TicketServer.class and java.policy (but not the rmic-generated classes) to another directory and try running it from there to check that it is downloading the client stub using the server's code base (e.g. remove the stub from the web directory and check that it fails!).

Note that the server (in another process - potentially on another computer) prints messages in response to your running the client. Viva Distributed Objects.

When you have finished remember to kill the server and then the rmiregistry (e.g. using Ctrl-C, or kill (UNIX) or the Task Manager (Windows)).

11. Additional notes - links to TCP/IP operation

Here is a tcpdump of the packets exchanged running this example: ../rmi-packet-trace.txt

This figure shows the various ports and connection in the example: ../packet-trace-pic.pdf

12. See Also

Sun's own Java RMI documentation and tutorials, at least some of which is included with the Java SDK.
Also try their web site.

EOF